COMP 303 Term Project Fall 2016
Single Cycle Processor Design

Deadline: Jan 6th, 2017 midnight

Instructor: Didem Unat
Corresponding TAs: Mohammad Laghari (ENG 110), Najeeb Ahmad (ENG 110)

Notes:

- You may do the project as a team of 2.
- You may discuss the problems with your peers but the submitted work must be your own work.
- No late assignment will be accepted.
- Submit a SOFT copy of your assignment to the blackboard.
- This assignment is worth 10% of your total grade.
- You will be scheduled to do a **demo t**o explain your design and implementations to TAs.
- You may use the Blackboard **forum** to post questions regarding the project. Feel free to answer your peer's questions in the forum.
- TAs will hold two **PS hours** on 21st and 27th. Make sure to get familiar with the processor simulator before coming to the PS hour.
- TAs are here to help you, but you should also keep in mind that they too are students and would be having their final exams and project deadlines. You should utilize their office hours and PS hours completely.
- Read the whole project description before starting.

Academic Integrity: You may consult any of the MIPS architecture documentation available to you in order to learn about the instruction set, what each instruction does, etc. But we expect your design to be entirely your own. If you are unsure if it is okay to borrow from some other source, just ask the TAs, and give credit in your final write-up. If you are unsure about asking the TAs, then it is probably not okay. Plagiarism in any form will not be tolerated.

Project Description

In this project you will design and implement a 32-bit processor using Logisim. Logisim is a software logic simulator which provides basic components to build a simple processor. You will first design an ALU and use it in your processor.

Part 0: Preamble

You need to setup the environment first in order to run the simulator.

1. Make sure you have the latest distribution of JRE
2. Download Logisim simulator: http://www.cburch.com/logisim/
3. Download the cs316.jar (you will use this while implementing the Single Cycle Datapath)
4. Download the 1-bit adder provided on blackboard

Part 1: ALU Design

You all know by now that the arithmetic logic unit (ALU) is the most crucial part of a processor. You have to design the ALU that you would later use in your single-cycle datapath. You are supposed to design a 32-bit ALU.

Your ALU will support some operations from MIPS but not all the MIPS instructions. The table below shows the instructions which your ALU should support and their corresponding opcodes. Note that these opcodes are the same as the ones in MIPS. When implementing the ALU, **you ARE NOT allowed to use the Logisim arithmetic library. You are supposed to build your own ALU.**

OPCODE	Instruction	Operation
000000	sub	C = A - B
000001	add	C = A + B
000010	mult	C = A * B
000011	and	C = A & B
000100	or	C = A | B
000101	xor	C = A ^ B
000110	sll (shift left logical)	C = A << B
000111	slt (set less than)	C = 0 IF (A >= B) ; 1 IF (A < B)
001000	sra (shift right arithmetic)	C = A >>> B
001001	srl (shift right logical)	C = A >> B

NOTE: You must have the following input/output signals in your ALU;
Input: Operand A (32 bit), Operand B (32 bit), Operation (6 bit)
Output: Carryout (1 bit), Overflow (1 bit), Zero (1 bit), C (32 bits)
Your ALU should have the following orientation

1. Two input pins (32 bit wide) on the **WEST** side
2. An opcode pin (6 bit wide) on the **SOUTH** side
3. An output pin (32 bit wide) on the **EAST** side

Part 2: Single Cycle Processor Design

In this part, you will use the ALU designed in Part 1, components available in Logisim and the library we provide you to develop a single-cycle processor. You may use any of the components available in Logisim for this part except for the ALU. Use the ALU you have designed in Part I for Part II.

Apart from the ALU instructions described in Part 1, your processor should include the following load/store, branch and jump instructions and a custom instruction that you design. Make sure to discuss the custom instruction in your report. Be creative!

OPCODE	Instruction	Operation
001010	lw	Load word from memory into register
001011	sw	Store word from register to memory
001100	beq	Branch to a label if two registers are equal
001101	bne	Branch to a label if two registers are not equal
001110	j	Unconditional branch
001111	myIns	*Use your imagination (custom instruction)*

Register File: You are provided with the register file to use in the design. The name of the register file is cs316.jar, you have to import it to your workspace as a separate library (Project -> Load Library -> Jar Library...). When you have loaded it, the program will ask you for a class name, make sure you enter this (edu.cornell.cs316.Components) as the class name. Otherwise the library will not be included.

The register file library contains a 32-bit wide by 32- registers deep register file. Register $0 is hard-wired to zero at all times, and writes to $0 are ignored. Inputs *rA* and *rB* are used to select register values to output on the 32 bit *A* and *B* outputs. On the rising edge of the clock, if *WE* is high (write-enable signal is set), the data value at input *W* is stored in register *rW*. As you expect, *rA, rB* and *rW* are 5 bits.

Memory: For the memory part of your design, please note that the instruction memory can be built using either a ROM or a RAM module. The data memory, however, has to be a RAM module.

Part 3: Testing your processor

To test your processor, you will need to

- Write an assembly program in a file named *test_program_asm.txt* (pseudocode for the program given below)
- Provide corresponding machine instructions for your assembly program in a separate file named *test_program_code.txt*. These instructions represented in hexadecimal (not in binary.)

We are providing you with example *test_program_asm.txt* and *test_program_code.txt* files to help you write you own. Please note that you **cannot** use these files directly to test your implementation as the opcodes for your implementation may differ from the given ones. We are also providing you *test_program_data.txt* file that you will use to test your implementation. Below is the brief description of the example files.

test_program_asm.txt contains assembly instructions and corresponding machine codes for a simple bank transactions simulator program. For instance, the first assembly instruction in the program is *addi $5, $0, 1* followed by the line 0: 20050001, where **0:** shows address of the instruction in memory and 20050001 shows instruction machine code. The purpose of this file show correspondence between assembly instructions and their machine code while also showing their address in memory. To create this file, you will follow these steps:

- Write assembly instructions for your program as per the pseudocode given below
- Generate hexadecimal machine code for the assembly instructions and write below each assembly instruction along with the memory address where the instruction will be placed (as per the sample *test_program_asm.txt* file provided). As you don't have an assembler for your CPU (assembler converts assembly instructions to machine code), you need to manually convert assembly instructions to machine code. The reason why haven't provided you with this file is because your machine instructions may differ depending upon the register addresses you choose in your implementation.

test_program_code.txt contains only the machine code instructions of your program. You can create this file by simply placing machine code (without address part) part of your *test_program_asm.txt* file into *test_program_code.txt* as per the given format. This file will be used to load instructions into instruction memory of your processor for testing. As you will see, Logisim allows instructions and data to be loaded into memory by right clicking on memory module and assigning a file to the memory using **Load Image** option. We will use this option to load *test_program_code.txt* file into instruction memory of your CPU.

test_program_data.txt file contains data for your program. This file will be loaded into data memory of the processor using the **Load Image** option described above to test your implementation. Please note that the first data element will loaded at data memory address 0, the second at address 4 and so on.

.

We will use the data and code files to test your processor implementation in the demo.

- We ask you to implement the following program to test your implementation.
- In addition you should device a simple program to test your custom instruction you developed in Part II. Make sure to provide its code and data files as well in your submission.

Test code Pseudocode:

```
     Set result register to zero (say register r3)
Label:
     load first word from memory into register (register r1)
     load second word from next memory loc into register (register
r2)
     Multiply r1 and r2 and store result in r2
     Add r2 to r3
     If r3 > 6000
          Jump Exit
     Jump Label

Exit:
```

Here is a sample C implementation of the pseudocode:

C code:

```
     void main()
     {
         int sum = 0, count = 0;
         int num_array[] = {10, 9, 8, 700, 5, 6, 400, 1, 2, 3};

         while(1)
         {
             num1 = num_array[count++];
             num2 = num_array[count++];
             sum = sum + num1 * num2;
             if(sum > 6000)
                break;
         }
     }
```

Submission:

You are supposed to submit the following files for Part 1.

1. All components and any intermediate files made while implementing the ALU or its components (these files should be in .circ format)
2. The final ALU in one single file (.circ format)

You are supposed to submit the following files for Part 2.

1. The final processor design file (.circ format)
2. Report containing a circuit schematic
3. Table summarizing control signals
4. Description of your design and custom instruction.

You are supposed to submit the following files for Part 3.

1. Test programs: their assembly codes and instruction files
2. Trace of the test programs, demonstrating that correct result was obtained.

Grading:

- Part 1: 30 pts (each op is 3 pts)
- Part 2: 45 pts (each op is 7 pts, 10 pts for custom instruction)
- Part 3: 15 pts (10 pts for test code + 5 pts custom inst. test code)
- Report: 10 pts

References:

- [1] http://www.cs.cornell.edu/courses/cs316/2006fa/projects123.php
- [2] https://github.com/lightninglu10/singleCycle

You are provided with a sample project which is 16 bit in [2]. You can use it to get an idea of what your project will look like. But keep in mind that we are providing you with a 16 bit implementation which uses library components of Logisim, however, in your project, you are supposed to implement a 32-bit processor **without** using any of the library components.

GOOD LUCK.